STI Projet 2 - Étude des menaces

Authors: Besseau Thibaud & Rashiti Labinot Date: 2018-19-12

Introduction

Ce document a pour but de résumer les travaux effectués dans le cadre de la partie 2 du projet, à savoir :

- Reprendre un site web réalisé dans le projet pour la première partie
- Effectuer une analyse sur le fonctionnement et la sécurité du site web
- Identifier les vulnérabilités et les menaces
- Corriger et documenter ces vulnérabilités

La première partie du projet consistait à réaliser une application web fonctionnelle. Cette deuxième partie consiste donc à la sécuriser.

Description du système

Afin de réaliser notre application web, nous devions respecter certains points notamment l'utilisation de PHP version 5.3 ainsi que SQLite. Ces contraintes ont pour but de montrer certaines failles de sécurités dans ces versions et donc de les corriger par nos soins. Aucune autre technologie n'est autorisée.

Partie WEB

Notre site web se trouve dans le répertoire "html" qui contient la page index et plusieurs répertoires dont :

- public pour les fichiers en rapport avec CSS et javascript
- config pour les fichiers en rapport avec la base de données (fonctions manipulant la base de données)
- src pour les fichiers représentant les différentes pages de notre site

Étant donné que toutes nos pages du site web se trouve dans le répertoire src, nous avons donc les descriptions suivantes :

Nom du fichier	Description
compose.php	Formulaire permettant d'envoyer un message
delete.php	Fichier permettant de supprimer un message
delete_user.php	Fichier permettant de supprimer un utilisateur
lock_unlock.php	Fichier permettant de suspendre/activer un utilisateur
login.php	Formulaire permettant de s'authentifier sur le site web
logout.php	Fichier permettant de se déconnecter du site web
mailbox.php	Page affichant l'intégralité des messages destinés à l'utilisateur
password.php	Formulaire permettant de changer le mot de passe utilisateur
profile_user.php	Page affichant le profil de l'utilisateur et ses informations
read.php	Page affichant le message reçu pour l'utilisateur
send.php	Fichier permettant l'envoi du message de l'utilisateur
users.php	Page affichant la liste des utilisateurs pour l'administrateur

Partie Base de données

Périmètre de sécurisation

Notre périmètre de sécurisation est donné par le cahier des charges. Elle se limite donc uniquement à l'application web, ce qui comprend donc les différents fichiers et fonctionnalités déployés via Apache ou SQLite.

La sécurité concernant le réseau qui entoure l'application ainsi que la sécurisation des machines physiques et la sécurisation des langages/librairies ne sont pas pris en compte ici.

Sources de menaces

Nous avons regroupé les sources de menaces en plusieurs catégories :

- Utilisateurs malins
  - Probabilité : Haute
  - Motivation : Avoir plus de privilèges, lire les messages des autres
  - Cible : Les crédentials des utilisateurs et des administrateurs
- Éventuels concurrents
  - Probabilité : Moyenne
  - Motivation : Rendre l'application inutilisable, copier la structure du site web
  - Cible : Le fonctionnement de l'application
- Hacker, script-kiddies
  - Probabilité : Moyenne
  - Motivation : Gain de reconnaissance, amusement
  - Cible : L'entierté de l'application web
- Cybercriminels
  - Probabilité : Faible
  - Motivation : Récupérer des mots de passe, des emails et utiliser le site web comme redirecteur vers des autres sites malveillants
  - Cible : Données de l'utilisateur
- Organisation étatique
  - Probabilité : Très faible
  - Motivation : Récolter des données pour de l'analyse/espionnage
  - Cible : L'entierté de l'application web

Scénarios d'attaques

Le but de ce point est de définir plusieurs scénarios d'attaques qui pourraient mettre à mal notre application web.

Chaque scénario comportera une description, un niveau d'importance, une source, une motivation, une cible et un contrôle (contre-mesure).

Vol de données sensibles

Titre	Description
Scénario	Un utilisateur curieux aimerait lire les messages entre deux collègues car ils ne s'entendent pas bien
Impact	Haut
Source de la menace	Script Kiddies, Hackers, utilisateurs malins
Motivation	Gloire (script kiddies, hacker), argent (cybercriminels), fierté (utilisateurs malins)
Cible	Données utilisateurs (mots de passe, rôles, messages)
Contrôles	Cacher le contenu de la base de données en vérifiant les entrées des utilisateurs

1. Injection SQL

L'injection SQL est une attaque classique et très basique. Il est l'attaque la plus utilisée selon le site de l'OWASP. Notre site web est touché par ce genre d'attaque à cause des différences champs textes que l'utilisateur peut remplir.

2. Mots de passe en clair dans la base de données

L'autre point sensible dans les bases de données est le stockage des mots de passe. Si un attaquant trouve un accès à la base de donnée par mégarde alors il aura accès au mot de passe en clair.

Il est fortement recommandé de chiffrer les mots de passe et les enregitrer chiffrés. Cela permettra de cacher les mots de passe même si la base de données est compromise.

Contournement d'authentification

Titre	Description
Scénario	Un utilisateur arrogant veut montrer à tous ses collègues de quoi il est capable. Pour prouver que c'est lui le meilleur en informatique, il va se connecter en tant qu'administrateur sans connaître le bon mot de passe
Impact	Haut
Source de la menace	Script Kiddies, Hackers, utilisateurs malins
Motivation	Gloire (script kiddies, hacker), argent (cybercriminels), fierté (utilisateurs malins)
Cible	Données utilisateurs (mots de passe, rôles, messages)
Contrôles	Cacher le contenu de la base de données en vérifiant les entrées des utilisateurs

1. Mot de passe faible

Le mot de passe doit toujours répondre à un certain critère comme sa complexité sur le nombre de caractères, l'utilisation de majuscule, caractères spéciaux ou chiffres. Tous les utilisateurs devraient respecter ces critères mais surtout l'administrateur car il est le point sensible de toute l'application. L'administrateur peut tout faire sur le site web, le perdre serait catastrophique.

Contre-mesures

Buffers overflow

Normalement, les problèmes de type buffer over flow ne sont pas causé directement par les langages interprétés comme PHP mais vu que le moteur PHP est écrit en C, il est quand même possible d'exploiter des failles de types buffer overflow dans le moteur PHP. Les buffers over flow en PHP peuvent permettre d'exécuter du code de manière arbitraire. Vu que cela se passe dans le code C du moteur PHP, il est très dur de déceler si notre code peut permettre exploiter des vulnérabilités de ce type. Mais comme dit le dicton : il vaut mieux prévenir que guérir nous avons vérifié que toutes nos entrés soit plus petites ou égales à la taille de leurs champs dans la base de données via la fonction `strlen()` . Une autre solution pour contrer les buffers overflow serait d'utiliser des librairies comme `Suhosin` qui change la manière dont la mémoire est allouée ce qui permet de diminuer très fortement les attaques de types buffers overflow Source : `https://www.phpclasses.org/blog/post/338-6-Common-PHP-Security-Issues-And-Their-Remedies.html`

Proctection des mots de passes de la base de données

Les mots de passes stockés dans notre base de données étant des informations extrêmement sensibles car la plupart des utilisateurs réutilisent la même combinaison d'email et de mot de passe sur plusieurs sites. Afin de garantir à nos utilisateurs, que même si notre base de données était dumpé par des attaquants, leurs identifiants utilisés sur d'autres sites seraient toujours sûrs, nous avons décidé de ne plus stocker leurs mots de passe en clair mais de simplement stocker le hash de leur mot de passe.

La fonction `password_hash()` étant une fonction mathématique à sens unique, il est normalement impossible à partir d'un hash d'obtenir le mot de passe d'origine sans brute forcer la fonction de hashage pour trouver une correspondance. De plus, afin de se prémunir contre le brute force de nos hashs de mots de passe, nous avons rajouté du sel. Le sel consiste à rajouter à la suite du mot de passe de chaque utilisateur, une chaine de caractères aléatoire. Ainsi, chaque hash généré est différent des autres même si les mots de passes entrés par les utilisateurs sont les mêmes. L'utilisation de sel oblige donc un attaquant à brute forcer chaque hash un par un. Lors de l'utilisation de la fonction `password_hash()` un sel unique est généré défaut pour chaque mot de passe. Pour verifier les mots de passe saisis, il suffit de donner le mot de passe en clair et le hash à la fonction `password_verify` et celle-ci s'occupe d'appliquer le meme sel et de hasher le mot de passe de l'utilisateur pour le comparer au hash stocker dans la base de données.

Force des mots de passe

Il ne sert à rien de protéger les mots de passe stockés dans la base de données s'ils sont facilement brute forcable. Afin de se prémunir contre ce risque, nous avons décidé de forcer nos utilisateurs à choisir des mots de passes complexes. Sur notre site, les mots de passe doivent posséder les caractéristiques suivantes pour être valide:

Au moins 12 caractères alphanumériques + au moins un caractère spécial. Pour vérifier les mots de passes des utilisateurs, nous avons utilisé la regex suivante:

```
if (preg_match("/[^(?=\S{12,})(?=\S*[\W])(?=\S*[a-z])(?=\S*[A-Z])(?=\S*[\d])$]/",$password))
{
  //change le mot de passe
  ...
}
else
{
  echo("Votre mot de passe n'a pas la compléxité requise.
        Veuillez en resaisir un autre qui respecte les règles suivantes: \n
        12 caractères \n
        Au moins un caractère minuscule \n
        Au moins un caractère majuscule \n
        Au moins un chiffre \n
        Au moins un caractère spécial");
}
```

Ci dessous se trouve l'explication de notre regex:

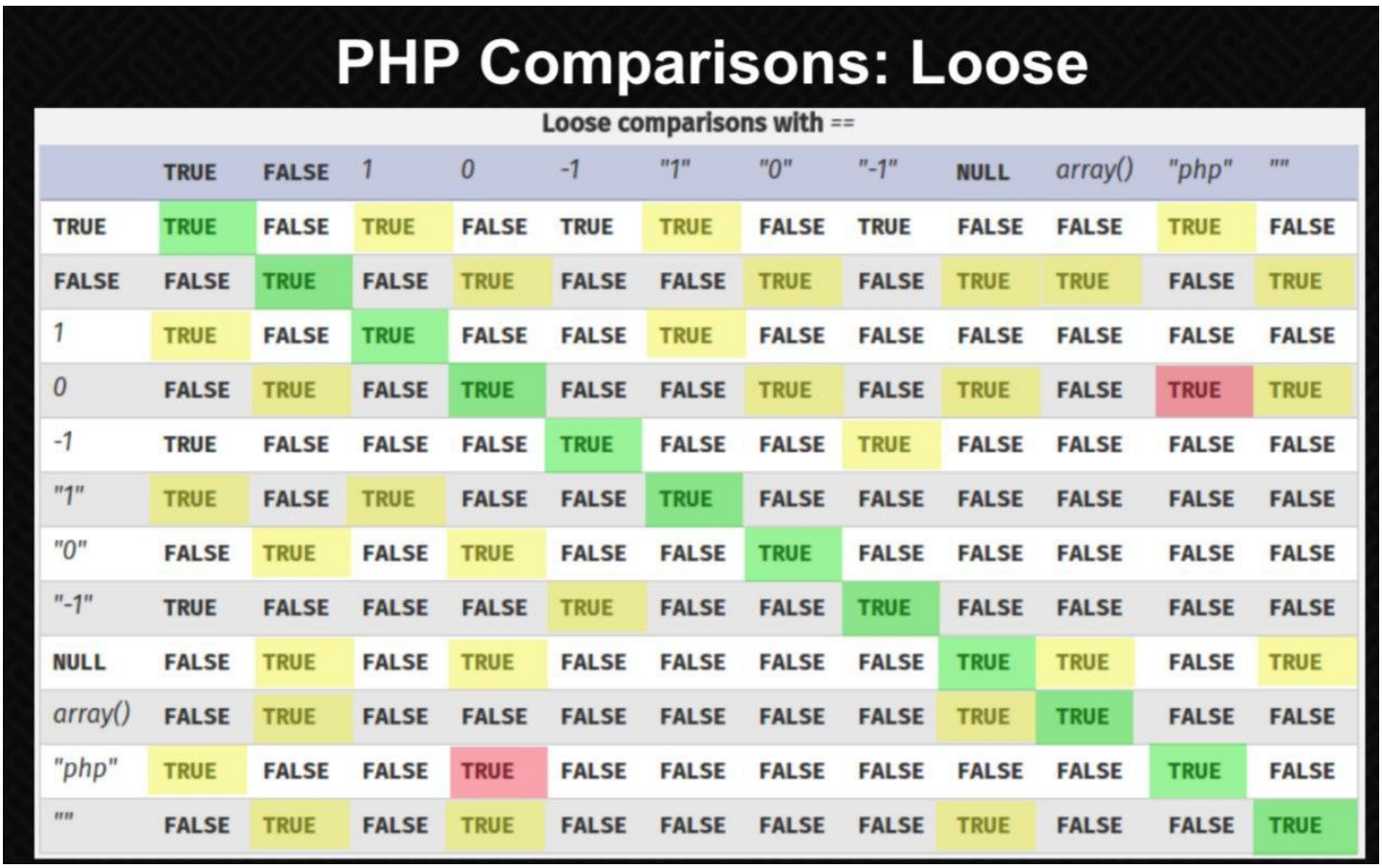
- `^` définit le début du mot de passe
- `(?=\S{12,})` définit que la taille du mot de passe doit être de 12 caractères au minimum
- `(?=\S*[\W])` définit que le mot de passe doit comporter au moins un caractère spécial
- `(?=\S*[a-z])` définit le mot de passe doit comporter au moins un caractère en minuscule
- `(?=\S*[A-Z])` définit le mot de passe doit comporter au moins un caractère en majuscule
- `(?=\S*[\d])` définit le mot de passe doit comporter au moins un chiffre
- `$` définit la fin du mot de passe

Protection des entrées

Une des attaques les plus communes sur les sites web est le cross site Scripting. Avec cette attaque, un attaquant peut par l'intermédiaire d'un site web faire exécuter du code (javascript par exemple) à certains clients. Il pourrait demander à un utilisateur de saisir ses identifiants et envoyer ces infos sur un site externe lui appartenant. Afin de ce protéger de ça nous avons échapper tous les caractères spéciaux comme `»` . Ainsi, un utilisateur ciblé par une attaque de type XSS verra simplement l'exploit sous forme de texte au lieu que celui-ci soit exécuté. Pour protéger les entrées des utilisateurs, nous avons utilisé la fonction `filter_var()` de PHP avec l'option `FILTER_SANITIZE_STRING` .

Types Juggling

Les comparaisons (non strictes) en php sont connues pour pouvoir ammener à des résultats qui ne sont pas ceux attendus. (Voir le tableau ci-joint:



	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

Pour s'assurer d'avoir toujours le meme comportement lors des comparaisons, nous avons décidé de forcer les comparaisons strictes comme `===` à la place de `==` ou `!==` à la place de `!=` . Pour les comparaisons de strings, nous avons choisis d'utiliser `strcmp()` ou `password_verify()` lorsqu'il s'agit de comparer des hashs de mots de passes

Expiration des sessions

Afin d'éviter des sessions qui sont valides encore des années après leurs créations et que de potentiels attaquant est le temps de brute forcer les identifiants de sessions, nous avons décidé de limiter la durée de validité des sessions à 30min. De plus, pour éviter les attaques de type sessions fixation, nous avons décidé de changer les identifiants de sessions à chaque changement de page via la fonction ci-dessous:

```
function isLoginSessionExpired() {
        //test if the last activity was more 30 minutes ago
        if(isset($_SESSION['LAST_Activity']) && (time() - $_SESSION['LAST_Activity'] > 1800))
        {
            session_unset();
            session_destroy();
            session_write_close();
            setcookie(session_name(),'',0,'/');
            session_regenerate_id(true);
        }
        else
        {
            //update the session last activity
            $_SESSION['LAST_Activity']=time();
        }

        //regenerate the session id all the 30 minutes
        if(time() - $_SESSION['CREATED']>1800)
        {
            // session started more than 30 minutes ago
            session_regenerate_id(true); // change session ID and invalidate old session ID
            $_SESSION['CREATED'] = time();
        }
}
```

Protection contre les attaques CSRF

Les attaques de type Cross site Request forgery consite à faire éxécuté à une victime des actions à son insu (comme donner les privilèges administrateur au compte de l'attaquant). Afin de se prémunir contre ce genre d'attaque, il est essentiel de s'assurer que quand un utilisateur va effectuer des modifications critiques (suppression de compte ou octroit de privilèges) se soit bien lui qui est initié cette action. Pour cela, il suffit de rajouter dans chaque requete `GET` ou `POST` un token unique appartenant à l'utilisateur. Ainsi si un attaquant effectue une attaque CSRF sur notre admin et essaye d'envoyer le formulaire pour modifier les droits de son compte sans que l'admin ne soit au courant, le formulaire ne sera pas acceptée par le site web car le token ne sera pas celui de l'admin.

Nous regenerons les tokens toutes les 10 minutes pour des raisons de sécurité, nous changons les tokens des utilisateurs. Les tokens ne sont pas générés via la fonction `uniqid()` car cette fonction est prédictible car elle se base sur le temps. A la place de cette fonction, nous préférons créer notre token avec la fonction `mcrypt_create_iv()` qui crée un vecteur d'initialisation vraiement aléatoire. Une fois le token crée, nous le stokons dans une variable de session:

Attention: La fonction `mcrypt_create_iv()` est dépréciée dans PHP 7 et doit être remplacée par la fonction `random_bytes()` qui existe depuis PHP 7.0

```
$_SESSION["token"] = bin2hex(mcrypt_create_iv(32, MCRYPT_DEV_URANDOM));
```

Pour chaque formulaire ou action, nous insérons le token dans l'URL pour les requêtes GET ou dans un champ caché pour les requêtes POST:

```
<input type="hidden" name="token" id="token" value="<?php echo $token; ?>" />
```

Puis sur la page PHP qui s'occupe de traiter la demande, nous vérifions que le token reçu est bien celui de l'utilisateur connecté:

```php
//pour les requêtes GET
if( $_SESSION ['token'] === $_GET ['token']) { }

//pour les requêtes POST
if( $_SESSION ['token'] === $_POST ['token']) { }
```

Si le site est accessible via le protocole HTTP, il est indispensable de changer le token régulièrement car il suffit à un attaquant de capturer le trafic entre la victime et le serveur web pour avoir le token. C'est pour cela qu'il est recommendé d'utiliser le protocle HTTPS pour chiffrer le trafic entre l'utilisateur et le site web.

Point de situation

Actuellement, nous avons contré les vulnérabilités suivantes:

Vulnérabilités	**Mesures de sécurités**
Controle d'accès	La gestion des accès via des cookies à été remplacé par des sessions PHP (plus sûr)
Controle des entrèes	Les entrées sont protégés et leurs tailles vérifiés
XSS réfléchi et stocké	Nous échappons les charactères spéciaux et aucun param GET n'est affiché
Injection SQL	Nous utilisons des requêtes préparées pour se prémunir de ce problème
Confidentialité des MDP	Les mots de passes stockés dans la base de données sont hashés et salés
Type juggling	Impossible car nous utilisons seulement des outils de comparaisons strictes
Session	Afin d'éviter le brute force des session, celle-ci sont fermé après 30 min d'inactivité
Session usurpation d'ID	Afin d'éviter l'usurpation, l'id de la session est regénéré régulièrement
Attaque CSRF	Pour chaque action, il est necessaire de soumettre un token qui est mis à jour réguièrement

A notre connaissance, il reste les failles suivantes

Vulnérabilités	**Mesures de sécurités à mettre en place**
RBAC sur la bd	Sur la base de données, il n'y pas de gestion des droits granulaire car seul un compte root est utilisé. Il faudrait mettre en place des droits en fonction des roles de chaque utilisateur ou service
Brute Force du login	Il est possible de brute forcer la page de login car il n'y a pas de captcha ou de banissement d'IP
Version de PHP obsolète	Depuis fin 2018, la version de PHP n'est plus en support EOL et donc elle ne reçoit plus de patch de sécurité
Version du serveur	Le serveur ubuntu hebergeant notre site est basé sur la version 14 dont le support étendue prendra fin à la mi avril. Après cette date, il n'y aura plus de patch de sécurité. Il faut donc plannifier une migration.
Trafic non chiffré	Afin de chiffrer notre trafic, il faudrait mettre en place un certificat SSL. Il est possible d'en générer [un gratuitement] (https://letsencrypt.org/). Nous avons mis en annexe un fichier docker compose qui se charge d'aller générer le certificat automatiquement (un nom de domaine et un email rattaché à ce nom de domaine sont necessaire

Le fichier DockerCompose devrait servir de base à tous déploiements de site web. Il propose met en place un NGNIX Proxy qui centralise toutes les requêtes des clients. Ce proxy est celui qui possède le certificat SSL. Ce certificat est mis à jour par un autre container qui s'occupe de vérifier périodiquement si le certificat est à jour et si besoin il le met à jour. Les autres containeurs sont plutôt basiques : un serveur NGNIX qui délivre le contenu au proxy et un containeur PHP. Pour finir il y a un containeur MYSQL qui ne peut être accédé que par le containeur PHP pour des raisons de sécurités.

Conclusion

Comme nous l'avons expliqué tout au long de ce rapport, sécuriser un site web PHP demande de prendre en comptes beaucoup de scénarios d'attaque afin d'être sûr d'avoir mis en place toutes les sécurités nécessaires. Il serait donc intéressant que tous développeurs fassent cette même analyse lorsqu'ils conçoivent le design de son site et à chaque changement ou ajout de fonctionnalités car la sécurité n'est pas un produit sur lequel on investit plusieurs heures de travail au début d'un projet mais un processus qui doit se poursuivre sur toutes la vie d'un site web.

Les technologies utilisées sont un autre point à prendre en compte lorsque l'on veut sécuriser une application (quelle soit web ou d'un autre type). Dans notre scénario, nous nous sommes acharnés à sécuriser une application sous PHP 5.6 mais cette version ne reçoit plus de correctif de sécurité depuis le début 2019. Il faut donc que chaque développeur s'il veut assurer la sécurité de son application gère le cycle de vie de son application (création maintenance régulière et décommissionnement). Dans notre cas, il faudrait migrer notre application sous une version long support de PHP7 si l'on voulait continuer à garantir la sécurité de notre application.

Le dernier point intéressant quand on développe des applications Web serait d'utiliser des Framework comme (Laravel ou Symfony) car la plupart des Framework intègrent par défaut des mesures de sécurités que la plupart des développeurs oublient de mettre en place (token CSRF par ex.).